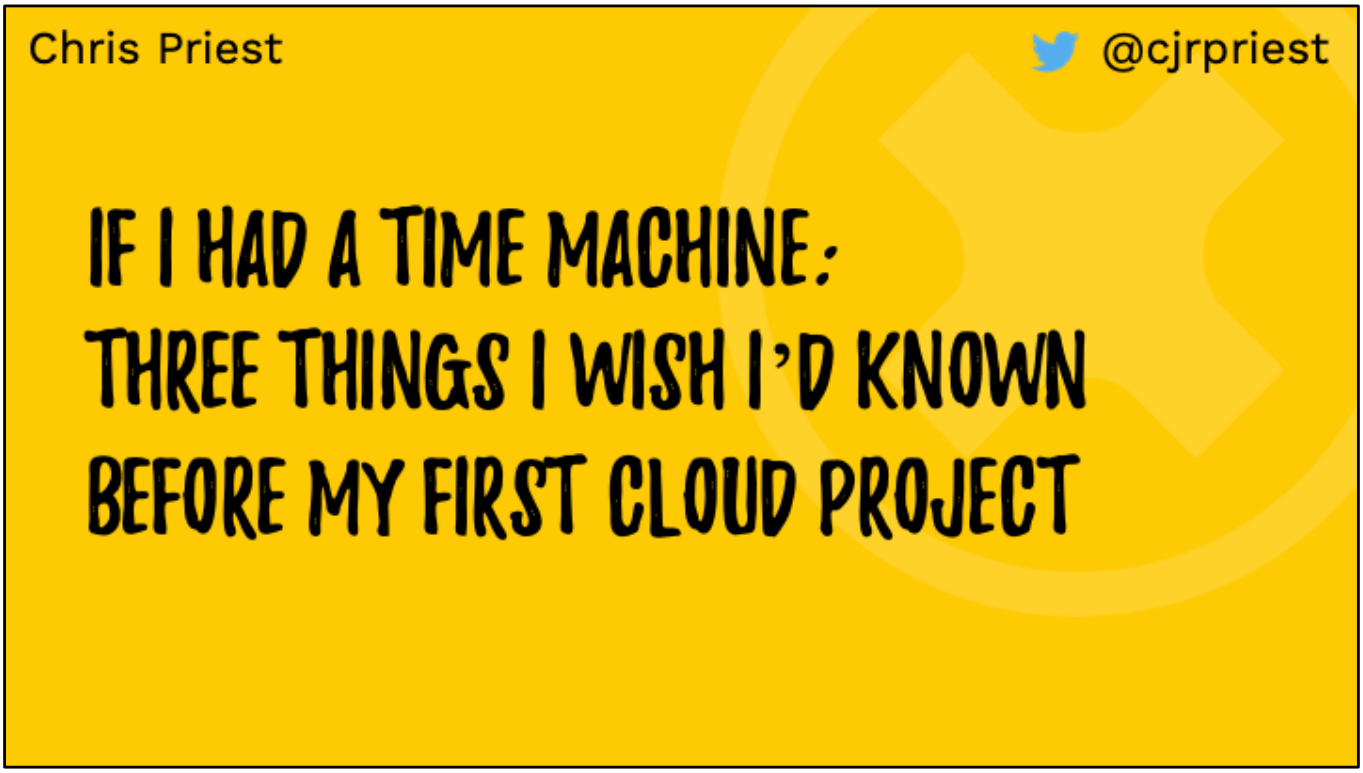


• Hello everyone, my name is Chris Priest and I'm here to tell you the three things that I would tell myself, before my first cloud project, if I had a time machine

• I've been involved in cloud projects in one way or another for nearly a decade

• and right now I work at the cloud-first consultancy Amido

• if you fancy working with a bunch of nice people exclusively on cloud projects then come & have a chat – we're downstairs

- I imagine that most of us have finished a project in the past (cloud or not) and then immediately said, with the benefit of hindsight, “I would have done that differently if I did it again”

- It’s not been any different for me in & around the cloud and

- So I’m gonna talk about three things
    - Serverless
    - Trade-offs in cloud compute
    - The cloud mindset

- If I had a time machine: I would tell myself what “serverless” *really* means
- I think the name is misleading, I think a better / more descriptive name is “utility compute”

I want you to think for a second about typical utility services you have in your home

Water, gas, electricity

The question is, What makes them utility services

- You dont care about how the service gets to you (how it's generated or refined), or about the underlying infrastructure (how it's pumped or transmitted to you) — that is all outsourced to the energy company, water company
- They are on demand & made available at precisely the point that you need them
- They are metered — you pay for exactly what you use (more or less, there might be a small service charge)

This is much like how serverless operates.

You don't worry about underlying infrastructure
Available on-demand at precisely the point it's needed
You pay for what you use

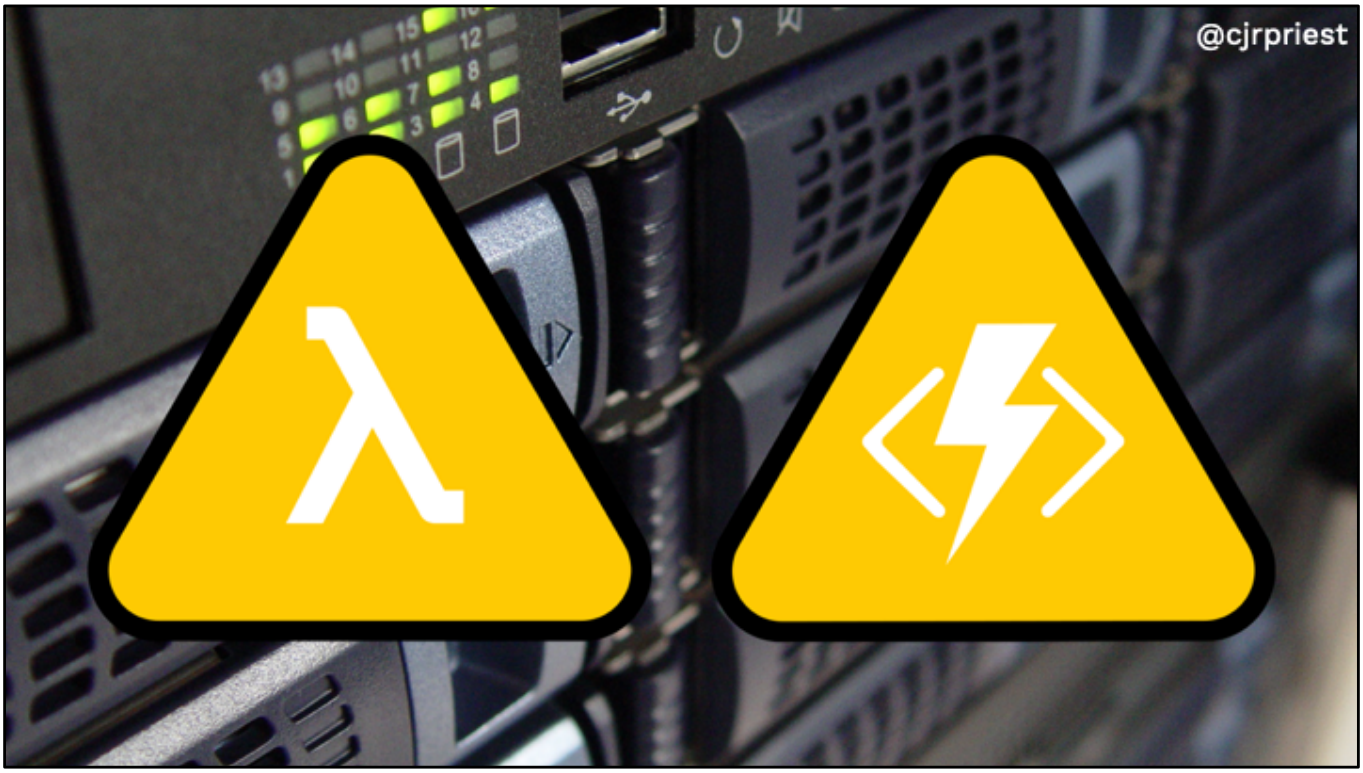
Conversely, Instance-based compute (AWS EC2, Azure VMs, CosmosDB, Azure App Services) are not utility computing / serverless

Just think about this for a second
Imagine that at the beginning of the day you buy a 1000 litres of water to use in your house – be conservative

- You Use what you use out of that metric tonne
  And then you dump what you've not used at midnight and then next day buy some more -- what a waste

- This is exactly how instance-based compute is bought
  - You pre-buy or pre-allocate some compute to use, and then do your best to make the most of it while you have it.
  - Nothing necessarily wrong with this, but it helps to understand the paradigm.
  - We'll talk more about this later

When you shift your thinking into this space, and understand what serverless really means, then your approach / when you decide to use it - changes



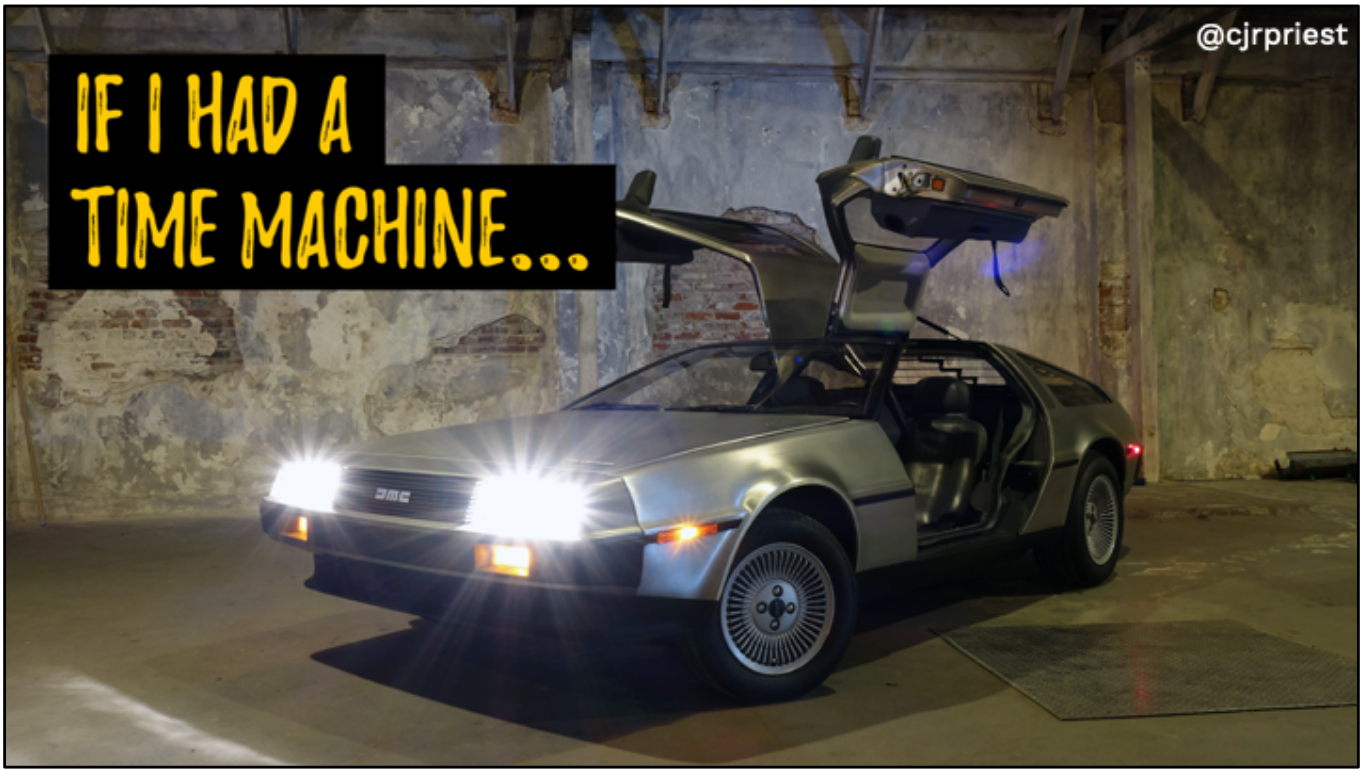
But beware! There are a few things that I wish I had known:

1) There are big differences depending on the choices you make...
    cold-start times — the time taken for your application to start in a serverless environment – depends on
        Language: C# Java take slightly longer than Node / Python
        other things, such as memory allocation, application size
    Not necessarily a problem, just worth being aware of

2) Some providers are better at scaling out than others
    For an in-depth view of this, then take a look at James Randall's excellent blog on how well Azure v AWS scale out under increasing load
     Just google Scaling Face Off
    another blog post following up where the problem is fixed
    Not necessarily a problem, just worth being aware of

3) The last thing to be aware of is a Shift in architecture or programming model, to an event-based approach
    serverless architectures tend to work best when they are event driven

if your serverless architecture is not based around events, then you might want to have a re-think



If I had a time machine: I would have taught myself the relationship between lead-time, operating-scale & cost
You see, Lead-time, operating-scale & cost are intrinsically related
But let's look at some definitions first

- Lead-time — the time it takes to react to a change in business or technical policy — for example, more visitors to your website, a technical failure, or a change to your infrastructure
  - A Low lead-time is measured in hundreds of milliseconds — typically serverless technologies
  - A High lead-time is measure in minutes - typically instance or managed app based technologies
- By Operating-scale — the scale at which you wish to operate your system
  - Low is measured in maybe tens or hundreds of concurrent users
  - High is measured in ten or hundreds of thousands of concurrent users
- Cost — the cost to run your site per user / interaction (not overall cost) — so the cost to service one customer, one software licence etc etc
  - Cost is very subjective and therefore very hard to set guidelines, but it must be proportional to your business.

e.g. high cost for a startup may be a low cost for an established business

So these three things are linked, and in a way that you might have heard some sales people refer to speed, quality & cost.
That is, you can only choose two.

There are three possibilities, and they are all good options

High operating scale + low lead-time = probably a serverless type of technology, and the cost is likely to be relatively high

High operating scale + low cost = probably an instance or managed app based technologies. But these have higher lead-time, or reaction time

Low lead-time + low cost = again probably a serverless type tech. Forced to keep operating-scale low

Why is this important to understand?
It's important, I think, so that you make the right choices at the moments in your project life cycle
You are going to want to make different choices if you are startup on a shoestring budget, or a prototype team in a large organisations
Vs a mature organisation with an established mature product

I was working at a start-up a few years ago, we had no money, no customers, we started with instance based compute

by doing this, we literally chose the worst possible option.
We chose the high cost, high lead-time option when we had a low operating scale.

The high-lead time mean our development cycle was slower & we were less able to react to the business.

The high costs meant our runway was shorter

there were literally no benefits to this approach
It would would have been much better for us if we'd

Starting with serverless based tech while we were at a low-operating scale

we would have had a low-lead time — meaning we could iterate rapidly with our software development, and we would have had the ability to

react if we suddenly achieved sales
low cost — would have meant that our runway would have lasted longer

as we matured (and we better understood our business) — it would have made sense to optimise towards a high-operating scale / low cost instance-based model, where we could more easily accept / manage the higher lead-time

The thing for me is that if you are in the early stages of a project or business, it's quite hard to justify NOT using severless techs

At a large operating scale, it’s hard to NOT justify instance based compute, based on the large percentage savings

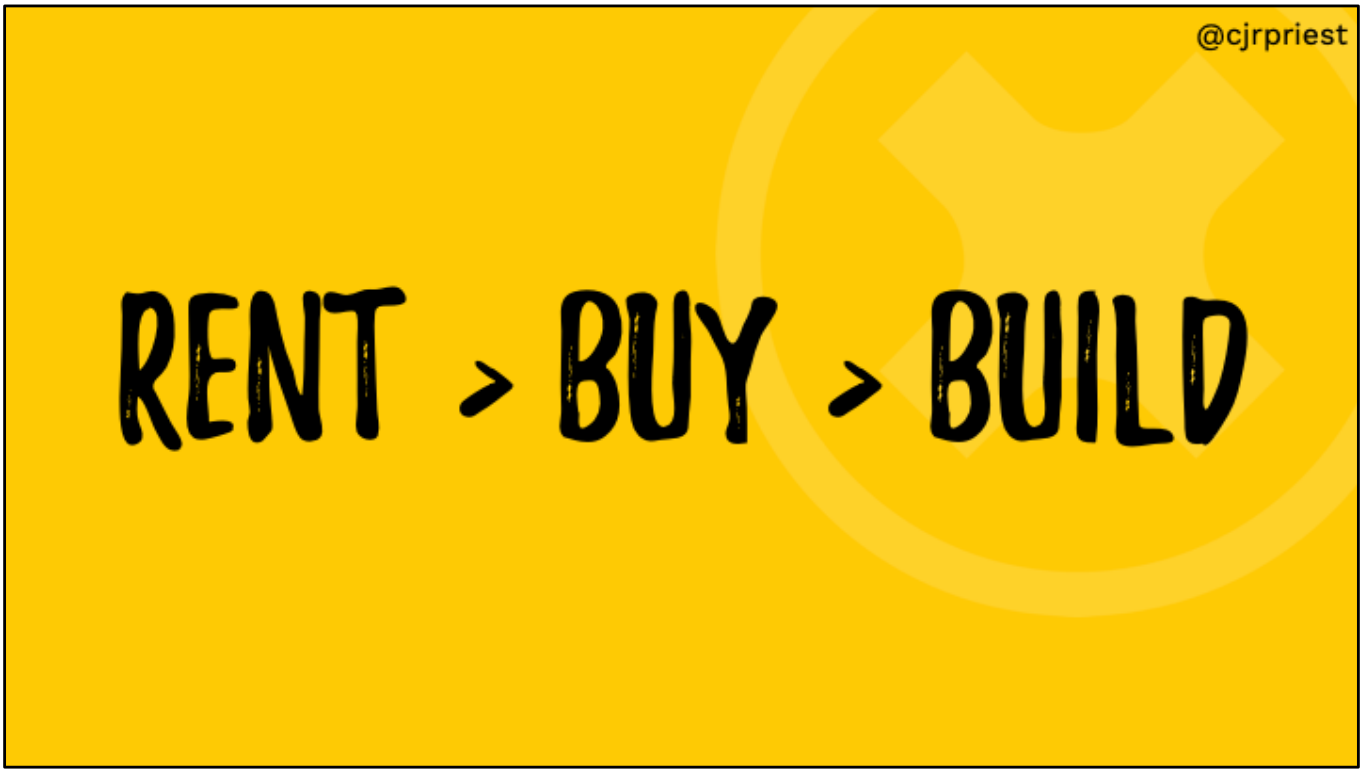
- Time machine image
- “If I had a time machine, I would convince myself to switch to the cloud mindset”
- And there are three areas here I want to talk about

“I would have told myself the cloud mindset is about”
... figuring what you *dont* need to do

deliver more, code less — use the amazing array of cloud services & sass products out there
This is likely to mean that you are solving a higher order problem

You are likely to find yourself piecing together the right combination of existing services, to make a sum that is greater than it’s parts
So instead of

- Storing files on disk — consider using Azure Blob Storage or AWS S3
- Using a database table as a queue (I know one of you has done that before) — consider Azure ServiceBus or AWS SQS
- writing your own facial recognition software — consider AWS Rekognition or Azure Cognitive Services Face API
- Re-inventing the wheel — RENT a wheel from someone else



Rent > buy > build

Consider renting (pay-per-transaction) over buying (paying for upfront perpetual license) over building (writing it yourself)

Writing it yourself is the least preferable option — it is likely to cost the most

“I would have told myself the cloud mindset is about”
...preparing for failure

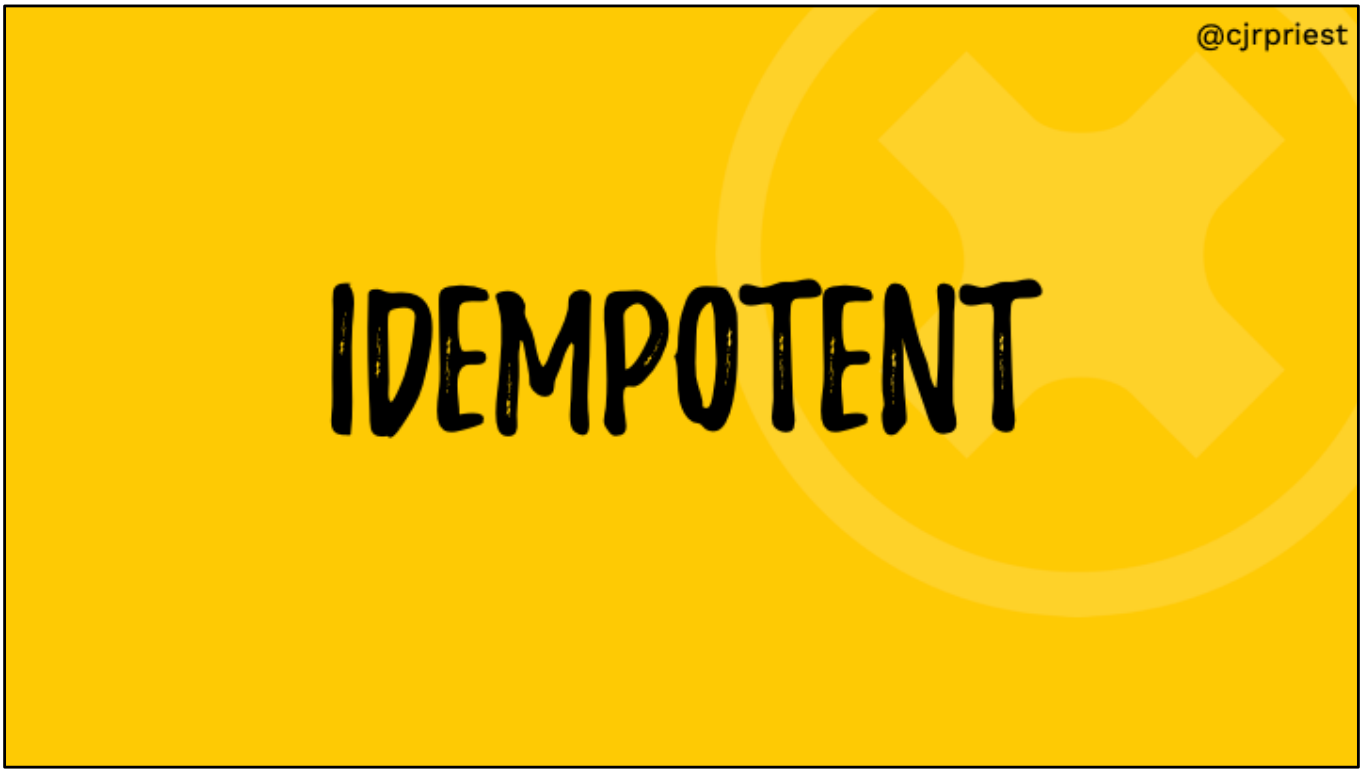
cloud systems are more often than not distributed systems — that is, they are not just built off of one application, they are constructed of many smaller systems, microservices, to create a sum that is greater than it’s parts
That means that you are going to need to traverse a network
The network is not guaranteed, you cannot guarantee that the network call that you made is going to succeed

It may not even be a transient problem — we were recently reminded on 4th September, by Azure, that as an industry, we’ve not yet cracked totally cracked HA — one problem in a data centre in the US had a global impact on many azure services

This sucks. But there are a couple of things to can do to make your life much easier

1) you can apply various connection management policies
The simplest of which is a retry policy
But there are other more complex or types, such as circuit breaker or retry with exponential backoff

There are some popular libraries that can help with this sort of thing — it's worth checking out Polly for .NET & Hystrix for Java — and there'll be equivalents for other languages



2) when you are designing your API, ensure it is idempotent. When you are calling another API, make sure that it is idempotent.

idempotency is a property of an API where it can be executed multiple times with the same input data without changing the result beyond the initial application

Why is this important?

> If you cannot guarantee the network, you need to be sure that multiple requests / retries can be made to an API with the same outcome

“I would have told myself the cloud mindset is about”

… know when NOT to use to cloud

We have this poster in our office — “Like cider, technology is better cloudy”
In general, I agree with this, but it’s fair to say that this isn’t always the case. It’s true that cloud is not the only way to do projects these days
story

I once worke d on a system that collected data from IoT devices in realtime, but then processed it offline

Crucially, there was NOT a requirement to complete the offline processing in a particularly timely fashion – a week was ok.

we started by designing & developing a solution for the cloud, but it quickly became clear that actually this wasn’t necessary — and that it would be significantly cheaper to run the process on-prem

Ok we would not be guaranteed the uptime, and we had a long

lead-time for infrastructure, but crucially if the system was down for a few days, we had a DR plan, and it didn't actually matter...

So when you've next another project on the horizon, watch this back online, and perhaps one or more of these things will be starting point for some interesting discussion

If you've got any questions, then please feel free to talk to me here at the conference, or send me a tweet!

Thank you!